Sprawozdanie nr 1
Praktyka i teoria szeregowania zadań

Data: 05.11.2020
Autor: Damian Tabaczyński
Indeks: inf136819
Grupa dziekańska: L6 (przeniesienie z L5)
Zajęcia: czwartek 11:45

1. Opis problemu

Problemem rozważanym w niniejszym sprawozdaniu jest problem szeregowania **n** zadań na jednej maszynie z minimalizacją łącznej sumy wag zadań spóźnionych $\sum w_j U_j$. Zagadnienie to można przedstawić również w notacji trójpolowej:

$$1 \mid r_j \mid w_j U_j$$

Dla każdego zadania **j-tego**, U_j wynosi 1 jeśli $C_j > d_j$ oraz 0 jeśli $C_j <= d_j$.

Ważną cechą tego konkretnego problemu jest niepodzielność zadań oraz wykonywanie zadań bez możliwości ich przerwania. Dodatkowo żadne zadanie nie może rozpocząć się przed swoim momentem gotowości $r_j <= C_j - p_j$.

Zgodnie z nomenklaturą podaną w zadaniu oraz przedstawioną na wykładzie:

- każde zadanie J_j opisane jest czasem trwania p_j, momentem gotowości r_j, oczekiwanym terminem zakończenia wykonywania d_j i wagą **w**
- C_j oznacza moment zakończenia wykonywania zadania J_j w uszeregowaniu

W ramach rozwiązania należy opracować algorytm listowy rozwiązujący przedstawiony problem, który nie przekracza złożoności obliczeniowej $O(n^2 \log n)$.
W sprawozdaniu w ramach ścisłości i zabezpieczając się przed złymi tłumaczeniami mogą zostać użyte pojęcia w wersji angielskiej.

2. Opis generatora instancji

Głównym założeniem generatora instancji jest to, że zawsze zwraca listę zadań, którą można ułożyć tak by suma wag zadań, które przekroczyły due date (d_j) wynosiła 0. W zamyśle sprowadza się to do rozpoczęcia generowanie mając już ułożoną sekwencje zadań. Taka metoda “nie wprost” zapewni, że zawsze sekwencje

można ułożyć w optymalny sposób. Dla każdego takiego zadania kolejno ustalamy pseudolosową wartość wagi z przedziału <1,10> oraz manipulujemy jego wartościami czasowymi (ready time oraz due date) by nakładały się i przeplatały w stosunku do wartości innych zadań. Taki zabieg uniemożliwi uszeregowanie zadań w łatwy sposób.

Zaimplementowany generator rozpoczyna przydzielanie wartości zadaniom grupami. Parametr wielkości grupy został ustalony na 10. Oznacza to, że algorytm wpierw wygenerował dane dla pierwszych dziesięciu zadań, następnie przeszedł do kolejnej dziesiątki itd. aż do wyczerpania liczby zadań. W każdej takiej grupie znajdują się zadania krótkie, średniej długości oraz długie, których łączna suma przetwarzania wynosi **liczebność grupy*10**. Oznacza to że dla grupy 10 łączny czas trwania wykonania tych zadań będzie wynosi 100 jednostek czasu. Przyjęto, że zadania krótkie stanowią 40% każdej grupy, średniej długości - 40% oraz długie 20% - przydział jest pseudolosowy. Zadania krótkie mają długość losowaną z przedziału <1,5> natomiast średniej długości z przedziału <6,15>. Zadania długie otrzymują swoje wartości jako ostatnie i są równe liczbie pozostałego czasu podzielonego przez liczbę zadań długich.

W drugiej fazie algorytmu dla każdego zadania jest ustalana wartości ready time, due date oraz waga. Dwie pierwsze charakterystyki ustalane są odpowiednio jako czas rozpoczęcia przetwarzania danego zadania w sekwencji ułożonej optymalnie oraz czas rozpoczęcia przetwarzania ($\mathbf{r}_j$) plus jego długość. Jednocześnie jest losowana wartość wagi z przedziału <1,10>.

Po uzyskaniu wszystkich właściwości następuje trzecia faza gdzie czasy ulegają modyfikacjom. Dla każdego zmniejszamy ready time o pseudolosową wartość z przedziału <0, $\mathbf{r}_j$>. Wartości due date zostają natomiast wydłużone tylko dla 50% zadań. Analogicznie ich zwiększanie odbywa się pseudolosowo z zakresu
<0, n*10-d_j> tak by nie przekroczyły maksymalnej wartości trwania całego szeregu czyli **n*10.**
Na koniec wszystkie zadania zostają wymieszane przy zapisie do pliku.

3. Opis algorytmu

Algorytm bazuje na rozwiązaniu listowym. Na początku sortowana jest lista zadań jednym z dwóch możliwych scenariuszy:

- sortowanie rosnąco według $\mathbf{d}_j$
- sortowanie malejąco według współczynniku $\mathbf{w}_j$ / $\mathbf{p}_j$

Lista dostępnych zadań - lista posortowanych uprzednio zadań według konkretnego scenariusza.

W przypadku sortowania został użytu algorytm Quicksort posiadający średnią złożonością **O(n log n)**, a najgorszą **O(n^2)**. Po wstępnym ułożeniu zadań następuje właściwe uszeregowanie:

Upływ czasu po wykonania zadania jest na bieżąco aktualizowany w trakcie działania algorytmu.

1. Jeśli lista dostępnych zadań jest pusta to przejdź do punktu 4. W przeciwnym wypadku przejdź do punktu 2.

2. Dopóki istnieje chociaż jedno dostępne zadanie to dla każdego dostępnego zadania w posortowanej wcześniej liście sprawdź w przedstawionej kolejności czy przekroczyło swój moment gotowości jeśli:

    a. Tak - sprawdź czy jest to zadanie spóźnione. Jeśli przekroczono jego czas $\mathbf{d_j}$ to jest dodawane do listy spóźnionych zadań i usuwane z listy dostępnych. W przeciwnym przypadku usuń je z listy dostępnych zadań oraz przejdź do punktu 3.

    b. Nie - sprawdź następne zadanie. Jeśli żadne zadanie nie jest gotowe to zwiększ aktualny czas by osiągnął poziom następnego gotowego zadania i przejdź ponownie do punktu 1.

3. Zadanie jest dodawane na koniec rozwiązania i następuje aktualizacja czasu po jego wykonaniu. Przejdź ponownie do punktu 1.

4. Dodaj do rozwiązania listę zadań spóźnionych oraz oblicz sumę ich wag.

Algorytm bagatelizuje zadania, które przekroczyły chociaż o jedną jednostkę swój due date, gdyż ich waga będzie wliczana do wartości kryterium oceny. Z tego powodu wszystkie już spóźnione zadania można uszeregować na sam koniec, gdy nie ma już żadnych dostępnych zadań nie-spóźnionych. Takie zachowanie nie wpłynie negatywnie na wynik a zdecydowanie poprawi działanie algorytmu.

Algorytm uruchamia dla każdej instancji obydwa scenariusze tzn. opisane wyżej postępowanie jest wykonywane najpierw z sortowaniem po $\mathbf{d_j}$ a potem po współczynniku $\mathbf{w_j / p_j}$. Złożoność jednego scenariusza wynosi **O(n log n + n^2)**. Pierwszy człon wyniki z sortowania Quicksortem, natomiast drugi z zagnieżdżonych dwóch pętli. Złożoność algorytmu wynosi **O(2n log n + 2n^2)**, co sprowadza się do złożoności **O(n^2)**.

4. Wyniki algorytmu i obserwacje

n	Wartość kryterium dla sztucznego pliku wynikowego **A**	Wartość kryterium uzyskana przez własny algorytm **B**	Względna różnica w [%] **C = (A-B)/A *100**	Czas działania algorytmu [ms]
50	246	15	93,9	95
100	486	1	99,79	93
150	724	0	100	96
200	950	5	99,47	97
250	1271	5	99,61	102
300	1596	0	100	98
350	2028	0	100	103
400	2232	14	99,37	104
450	2372	20	99,16	109
500	2564	12	99,53	110

Jak widać w przedstawionych danych algorytm osiąga kolosalne względne różnice dla wygenerowanych instancji. Pomimo zadowalających wyników algorytmu częściowy “sukces” jest spowodowany zbyt łatwym zadaniem szeregowania. Niestety w rezultacie niewystarczających zmian we właściwościach $\mathbf{d_j}$ oraz zbyt małej ingerencji w wartości losowane dla $\mathbf{r_j}$ instancje można zbyt łatwo uszeregować za pomocą sortowania zadań po rosnącym $\mathbf{d_j}$. Jest to jedna z przyczyn tak dobrych wyników.

Niemniej jednak algorytm w porównaniu do całej biblioteki instancji prezentuję się bardzo dobrze. Świadczy o tym średnia wartość różnicy kryteriów będąca na poziomie **67,48%.**